**StudySpace Design**

**Document**

Team 2

Version: 4.0

Date: 10/04/21

**Table of Contents**

1. Introduction

1.1 Purpose

1.2 Audience

1.3 Scope

1.3.1 Software Requirements

1.4 Feature Ranking

1.4.1 Must-Haves

1.4.2 Nice-to-Haves

1.5 Development Milestones

1.6 Definitions

1.7 Reference Materials

1. System Overview

2.1 Brief Description

2.2 Key System Components

2.2.1 Front-end

2.2.2 Back-end

2.3 Key System Functionalities

2.3.1 Setting Interests

2.3.2 Pages and Desks

2.3.3 User Matching

2.4 Key UI Components

2.4.1 Bars

2.4.2 Containers

2.4.3 Pages

2.4.4. Desks

2.4.4 Grids

1. Design Considerations

3.1 Assumptions and Dependencies

3.2 End-user Characteristics

3.3 General Constraints and Limitations

3.4 Goals and Guidelines

3.5 Development Methods

1. Architectural Strategies

4.1 Communication Mechanisms

4.2 Technology Specifications

4.2.1 Back-end

4.2.2 Front-end

4.2.3 Miscellaneous

4.3 Front-end Goals

4.4 Back-end Goals

4.5 Future Plans

4.6 User Interface Paradigms

4.7 External Database Management and Persistence

1. Design Decisions, Policies, and Tactics

5.1 Deployment

5.2 Data Storage

5.3 Coding Guidelines and Conventions

5.4 Ensuring Requirements Traceability

5.5 Software Testing

5.6 Software Maintenance

5.7 File Structure Organization

1. System Architecture

6.1 System Architecture Description

6.2 System Architecture Diagram

1. Architectural and Component-level Design

7.1 Class Diagram

7.1.1 Main Class

7.1.2 User Class

7.1.3 Moderator Class

7.1.4 Administrator Class

7.1.5 Friend Class

7.1.6 Group Class

7.1.7 Page Class

7.1.8 Desk Class

7.1.9 Post Class

7.1.10 Events Class

7.1.11 Chat Class

1. Data Design

8.1 Data Structures

8.1.1 Global data structure

8.1.2 Temporary data structure

8.1.3 Database description

8.2 Database Diagram

8.3 Database tables

8.3.1 user Table

8.3.2 group Table

8.3.3 user\_group Table

8.3.4 friend Table

8.3.5 chat Table

8.3.6 posts Table

8.3.7 response Table

1. User Interface Design

9.1 User Interface Design

9.1.1 Color Guidelines

9.1.2 Font Guidelines

9.2 User Interface Views

9.2.1 Landing Page (logged out)

9.2.2 Sign Up View

9.2.3 Home Page (Logged In)

9.2.4 Groups Home Page

9.2.5 Desk View

9.2.6 Posts View

9.2.7 User Profile Page

9.2.8 Profile Page (another user)

9.2.9 User Match View

1. Data Dictionary

**Revision History**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Date | Member | Version | Description | | | | |
| 3/6/2021 | Ann Baldonasa | Version 1.0 | Created and organized Design document outline | | | | |
| 3/16/2021 | Ann Baldonasa | Version 1.0 | Authored Sections 1 (Introduction), 2 (System Overview), 4 (Architectural Strategies) | | | | |
| 3/16/2021 | Arvin Benipal | Version 1.0 | Created User Interface Design (section 9) | | | | |
| 3/16/2021 | Kevin Lin | Version 1.1 | Authored Section 8 (Data Design) | | | | |
| 3/16/2021 | Muhammad Hashir | Version 1.0 | Authored section 3(design considerations),5(Design Decisions),6,10(data dictionary) | | | | |
| 3/18/2021 | Janelle Tait | Version 1.0 | SQA | | | | |
| 3/18/2021 | Mackenzie van Zanden | Version 1.0 | SQA | | | | |
| 3/18/2021 | Dayton Talarico | Version 1.0 | SQA | | | | |
| 3/19/2021 | Rohan Uddin | Version 1.1 | SQA | | | | |
| 3/18/2021 | Brian Carvalho | Version 1.0 | SQA | | | | |
| 3/18/2021 | Zeeshan Jafri | Version 1.0 | Created Class Diagrams and Authored section System Architecture and Component Level Design | | | | |
| 3/21/2021 | Ann Baldonasa | Version 2.0 | Made changes to sections 1, 2, 4 upon SQA suggestions | | | | |
| 3/22/2021 | Kevin Lin | Version 2.0 | Made changes to section 8 upon SQA suggestions | | | | |
| 3/22/2021 | Arvin Benipal | Version 2.0 | Made changes to UI design (section 9) upon SQA suggestions | | | | |
| 3/21/2021 | Muhammad Hashir | Version 2.0 | Made changes to section 3,5,6,10 upon SQA suggestions | | | | |
| 3/23/2021 | Janelle Tait | Version 2.0 | SQA | | | | |
| 3/23/2021 | Mackenzie van Zanden | Version 2.0 | SQA | | | | |
| 3/23/2021 | Dayton Talarico | Version 2.0 | SQA | | | | |
| 3/24/2021 | Rohan Uddin | Version 2.1 | SQA | | | | |
| 3/23/2021 | Brian Carvalho | Version 2.0 | SQA | | | | |
| 3/24/2021 | Zeeshan Jafri | Version 2.0 | Made changes to Class Diagrams based on SQA, and to Section 7 | | | | |
| 3/24/2021 | Ann Baldonasa | Version 3.0 | Made changes to sections 1, 2, 4 upon SQA suggestions | | | | |
| 3/24/2021 | Muhammad Hashir | Version 3.0 | Made changes to sections 6 and 10 | | | | |
|  |  |  |  | | | | |
|  |  |  |  | | | | |
|  |  |  |  | | | | |
|  |  |  |  | | | | |
|  |  |  |  | | | | |
|  |  |  |  | | | | |
|  |  |  |  | | | | |
|  |  |  |  | | | | |

**List of Tables**

## **Section 1**

1.3.1. Software Requirements

1.5. Development Milestones

## **Section 9**

9.1.1. Color Guidelines

## **Section 10**

10. Data Dictionary

**List of Figures**

## **Section 6**

6.1. System Architecture Diagram

## **Section 7**

7.1. Class Diagram

## **Section 8**

8.2. Database Diagram

## **Section 9**

9.2.1. Landing Page (logged out)

9.2.2. Sign Up View

9.2.3. Home Page (Logged In)

9.2.4. Groups Home Page

9.2.5. Desk View

9.2.6. Posts View

9.2.7. User Profile Page

9.2.8. Profile Page (as another user)

9.2.9. User Match View

# **Introduction**

This section provides an overview of the StudySpace Software Design Document.

## Purpose

* To provide a general description of the StudySpace software architecture.
* To describe the functional structure, data, and algorithms to be implemented.
* To describe the behavioral requirements.
* To identify the required system resources needed by the system’s software developers.
* To demonstrate the means to fulfil the requirements as specified in the StudySpace Software Requirements Specifications (SRS) document.
* To serve as a reference and support for the software developers in the creation of the StudySpace web application.

## Audience

The primary intended audience of this document is current and future developers tasked with the creating, testing, and maintenance of the StudySpace web application.

The secondary intended audience of this document is the project client, Mr. David Brown.

## Scope

### Software Requirements

|  |  |
| --- | --- |
| **Functional Requirement 1** | **User Profile Page** |
| Priority | High |
| Purpose | The software shall display information about the user. |
| Inputs | Related Interfaces  The Profile Page contains the following   * Interests * User Matches * Joined Groups * Past Posts * Friends |
| Operations | Required Processing   * The user must have already set their Interests so that the User Matches may be processed. |
| Output | Required Interfaces   * Joined Groups contains the links of the groups in which the user has joined. |

### 

|  |  |
| --- | --- |
| **Functional Requirement 2** | **Search Bar** |
| Priority | High |
| Purpose | The software shall allow the user to search for groups and users that the user is interested in. |
| Input | Related Interfaces  The Search Bar will consist of a text field and a Search button.  Valid Characters  To search for groups, the user will prefix the group with a hashtag.  Example: To search for groups related to CP164, the user will type #CP164.  To search for users, the user will prefix the user with an at sign.  Example: To search for a user with the username “bambi”, the user will type “@bambi”. |
| Operations | Required Processing   * Whether or not the user is searching for a group or a user must be determined.   Responses to Abnormal Conditions   * If there are no groups or accounts matching the user’s request, then the Search Results page will prompt a pop-up stating an Error Message. |
| Output | Related Interfaces   * Submitting your search request will result in a page displaying your search results. This page display is described in the Functional Requirement 3.   Error Messages   * If the user has searched for a group or user without the appropriate prefix (@ or #), then the user will get an Error Message stating, “Your search did not match any results. No users or groups found.” |

### 

|  |  |
| --- | --- |
| **Functional Requirement 3** | **Search Results** |
| Priority | High |
| Purpose | The software shall display the search results. |
| Input | Related Interfaces   * Search Results display will result in a list of groups or users depending on the search query. |
| Operations | Required Processing   * The user must have submitted a search query into the Search Bar. |
| Output | * If the user has searched for groups, then a list of groups along with their descriptions will be shown. * If the user has searched for a user, then a list of users matching the request will be shown. |

|  |  |
| --- | --- |
| **Functional Requirement 4** | **Home Page** |
| Priority | High |
| Purpose | The software shall show the user’s Home Page. |
| Input | Related Interfaces  The Home Page contains the following UI components.   * Main Navigation Bar * Sidebar * Search Bar * Top Groups Grid * Top Posts Grid |
| Operations | Validity Checks   * The user must have a valid account. * The user must have set their Interests so that their Top Groups suggestions may be displayed.   Responses to Abnormal Conditions   * If no Interests have been set, then the Top Groups Grid will be empty and there will be a message in this area stating, “Interests must be set to suggest Top Groups.” * If the user has not joined any groups, then the Joined Groups list on the Sidebar will not have any groups and there will be a message in this area stating, “You have not joined any groups.” * If there are no posts in their groups, then the Top Groups Grid will be empty and there will be a message in this area stating, “No posts found in your Top Groups.” |
| Output | Related Interfaces   * If the user has clicked the Search Bar, the Home Screen will expand the Search Bar. |

|  |  |
| --- | --- |
| **Functional Requirement 5** | **Groups Home Page** |
| Priority | High |
| Purpose | The software shall show a single Group Home Page. |
| Input | Related Interfaces  The Groups Home Page contains the following components   * Main Navigation Bar * Sidebar * Search Bar * Pages   + Posts   + Desks |
| Operations | Required Processing   * Display past posts made in this group. |
| Output | Timing Considerations   * When a Desk expires, the Desk must be immediately removed from the Page. |

## Feature Ranking

### Must-Haves

* User creation functionality
* Group creation functionality
* User Profile Page displaying the users:

1. Interests
2. User Matches
3. Joined Groups

* Home Page with:

1. Main Navigation Bar
2. Sidebar
3. Top Groups Grid
4. Top Posts Grid

* Groups Home Page with:

1. Main Navigation Bar, Sidebar, and Search Bar
2. Page Creation functionality
3. Desk Creation functionality
4. Post Creation functionality

* Friends Page display
* Message History Display

### Nice-to-Haves

* Landing Page with:
  + Sign-in functionality
  + Sign-up functionality
* User Profile Page with:
  + Setting Interests functionality
  + Setting Account Information functionality
* Adding Student Profile to Friends List functionality
* Removing Student Profile from Friends List functionality

## Development Milestones

|  |  |  |
| --- | --- | --- |
| **ID** | **Milestones** | **Date** |
| 1 | Organized front-end and back-end git repository structure | 3/26/21 |
| 2 | Set-up and configuration | 3/26/21 |
| 3 | Deployed Home Page with User endpoint (user icon and username) obtained from API. | 3/27/21 |
| 4 | Implemented user creation functionality | 3/28/21 |
| 5 | Implemented group creation functionality | 3/28/21 |
| 6 | Implemented User Matching functionality | 4/2/21 |
| 7 | Deployed user’s Profile Page with display of Interests, User Matches, and Joined Groups | 4/2/21 |
| 8 | Included Main Navigation Bar and Sidebar in Home Page | 4/4/21 |
| 9 | Included Top Groups Grid and Top Posts Grid in Home Page | 4/4/21 |
| 10 | Included Searching functionality and Search Bar in Home Page | 4/6/21 |
| 11 | Deployed Groups Home Page with Main Navigation Bar and Sidebar | 4/6/21 |
| 12 | Included Searching functionality and Search Bar in Groups Home Page | 4/8/21 |
| 13 | Included Page Creation functionality in Groups Home Page | 4/8/21 |
| 14 | Included Desk Creation functionality in Groups Home Page | 4/10/21 |
| 15 | Included Public and Private Desks functionality in Groups Home Page | 4/10/21 |
| 16 | Deployed Landing Page with authentication | 4/12/21 |
| 17 | Deployed User Profile Page | 4/12/21 |
| 18 | Included User Account Information Display in User Profile Page | TBD |
| 19 | Included Interests Display in User Profile Page | TBD |
| 20 | Included Edit-and-Save Interests functionality in user’s Profile Page | TBD |
| 21 | Added feature to be able to view Student Profile Page via the Search Bar, Main Navigation Bar, and Sidebar | TBD |
| 22 | Added feature to add a user as a friend. | TBD |
| 23 | Added feature to unfriend a user. | TBD |

## Definitions

1. **API**

Application programming interface (API).

1. **BACK-END**

The part of a software system that is not visible or accessible to a user of that system and processes all of the user’s requests via the front-end part of the application.

1. **CONFIGURATION AND SETUP**

Configuration and setup in the context of this project mean before proceeding to development and testing of features, developers must complete the following:

* Access and import the necessary packages and libraries.
* Create the necessary accounts on third-party websites that will be used in development.
* Devise and agree on a process for storing and differentiating between production and development variables.
* Ensure that all required software dependencies are installed.
* Agree on a file structure and software development workflow.

1. **DESK**

A Desk is a virtual space wherein students may collaborate. Each desk is temporary; it only exists as long as there are people using it. After everyone leaves the desk, it is removed from memory.

Each desk has an admin who can set a limit to the number of members and can add signifiers such as language, age category, and other factors.

1. **DIFFERENCE BETWEEN USER AND STUDENT**

Student is a way of referring to other users.

1. **FRONT-END**

The part of a software system that is visible or accessible to a user of that system and makes requests to the back-end part of the system.

1. **INTERESTS**

Interests refer to the parameters that the user has chosen and the ranking of these parameters. The choice and ranking of parameters are key to determining User Matches.

The user can only set 3 interests (parameters).

*Example.*

Alice has chosen the following parameters: Co-op, Friends, CP104, Programming and she has the following Interests:

* Co-op
* Friends
* CP104

1. **LANDING PAGE**

This is the page that users of the StudySpace web application will see after clicking the domain name.

1. **MUST-HAVES**

Must-Haves are features, functionalities, and pages that are prioritized and must be completed by the StudySpace team before the end of the course (April 12, 2021).

1. **NICE-TO-HAVES**

Nice-to-Haves describes a list of features, functionalities, and pages that are valuable but are not prioritized. These items do not need to be completed by the StudySpace team before the end of the course (April 12, 2021).

1. **POPULARITY**

In terms of groups, popularity is measured in the number of users.

1. **PROOF OF CONCEPT**

Proof concept (POC), also known as proof of principle, is a realization of a certain method or idea to demonstrate its feasibility or a demonstration in principle to verify that some concept or theory has practical potential. For example, one of the key goals in developing this application is realising the User Matching functionality.

1. **REACT COMPONENT**

React components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML via a render() function.

1. **REACT STATE**

React states are built-in objects inside React components. These states are where values related to that particular component are stored.

When the state (data) changes, the components that house the states are re-rendered to reflect the changes.

1. **RESTFUL API**

One way to implement an API is by using the REpresentational State Transfer (REST) architecture to handle front-end requests to the back-end. The REST pattern provides simple, uniform interfaces that can be used to make data, content, and other digital resources available through web URLs. This allows for the consumption of these resources through web and mobile applications.

1. **SINGLE-PAGE APPLICATION**

A single-page application (SPA) is a web application or website that interacts with the user by dynamically rewriting the current web page with new data from the webserver, instead of the default method of the browser loading an entirely new page.

1. **SIMILAR INTERESTS**

Users have Similar Interests if they have chosen at least 2 of the same parameters in their Interests.

*Example.*

Alice has the following Interests:

1. Co-op
2. Friends
3. CP104

Bob has the following Interests:

1. CP104
2. Programming
3. Co-op

Charlie has the following Interests:

1. Financial Markets
2. Co-op
3. Linear Algebra

In this case, Alice and Bob have at least 2 of the same parameters (Co-op, CP104) in their Interests and thus have Similar Interests so they would be included in each other’s User Matches. Charlie does not have enough Similar Interests with either Alice or Bob (Co-op) so Charlie would not be included in either Alice or Bob’s User Matches.

1. **STUDENT PROFILE**

The profiles of other users.

1. **TOP GROUPS**

The most popular groups that are similar to the user’s interests that the user has not yet joined.

*Example.*

If the user is interested in joining a group about co-op, there may be several groups named Co-opA and Co-opB that they are interested in potentially joining. However, if Co-opA has more users than Co-opB, then Co-opA will be featured in the user’s Top Suggested Groups shown on the Home Page.

*Example.*

The user is interested in groups related to CP104 and there are already a few groups called CP104\_A and CP104\_B. If the user has already joined CP104\_A, then CP104\_A will not appear in their suggested Top Groups and CP104\_B will appear in their suggested Top Groups as they have not joined CP104\_B. CP104\_A will instead appear in their Joined Groups list.

1. **UI**

User Interface.

1. **USER MATCHING**

User Matching is the process of determining compatibility with other users so that User Matches may be determined.

User Matching is a two-step process:

1. Determining if users have enough Similar Interests. *Refer to Similar Interests definition for more information.*
2. Comparing the priority and ranking of their parameters.

*Example.*

Alice has the following Interests:

1. Co-op
2. Friends
3. CP104

Bob has the following Interests:

1. CP104
2. Programming
3. Co-op

Charlie has the following Interests:

1. Financial Markets
2. Co-op
3. Linear Algebra

Dennis has the following Interests:

1. Co-op
2. Friends
3. Programming

Alice’s User Matches: Dennis, Bob.

* Charlie is not included in Alice’s User Matches because they have less than 3 similar interests.
* Dennis is higher on Alice’s User Matches than Bob because their first priority is the same (Co-op) while Bob’s first priority is CP104.

1. **USER PROFILE**

The profile of the user.

1. **USER ACCOUNT DETAILS**

User Account Details refers to the user’s account details such as username, password, and email address.

## Reference Materials

* [A Software Design Specification Template](https://www.bradapp.com/docs/sdd.html#SEC11)
* [SOFTWARE DESIGN SPECIFICATION TEMPLATE](http://www.cs.iit.edu/~oaldawud/CS487/project/software_design_specification.htm)
* [Software Design Document Template](https://docs.google.com/document/d/1pgMutdDasJb6eN6yK6M95JM8gQ16IKacxxhPXgeL9WY/edit#heading=h.wk232hcifrl1)
* [Software Design Document (SDD) Team O Event Driven DIS PDU Logger (EDDIS system)](http://portal.unimap.edu.my/portal/page/portal30/Lecture%20Notes/KEJURUTERAAN_KOMPUTER/Semester%202%20Sidang%20Akademik%2020112012/EKT420%20Software%20Engineering/Example%20of%20Software%20Design%20Document(SDD)/EDDISS.pdf)
* [IEEE Standard for Information Technology—Systems Design— Software Design Descriptions](https://perso.univ-st-etienne.fr/jacquene/gl/articles/IEEE-1016-2009.pdf)

# **System Overview**

StudySpace is a web application that provides students a digital space to interact with other like-minded students.

## Brief Description

At the highest level, the StudySpace web application is organized into key system functionalities, key system components, and key UI components.

* Key System Functionalities
  + Setting Interests
  + User Matching
  + Group Interactions
* Key System Components
  + Front-end
  + Back-end
* Key Visual Subcomponents
  + Main Navigation Bar
  + Sidebar
  + Search Bar
  + User Matches Container
  + Top Groups Grid
  + Pages
  + Desks

## Key System Components

### Front-end

The front-end has the following functions:

* Request data from the endpoints to populate subcomponents with appropriate data.
* Populate UI with data requested from the back-end.
* Assemble pages with the appropriate UI subcomponents and relevant data.

### Back-end

The back-end has the following functions:

* Get data requested by the front-end.
* Store data entered into user inputs into the database.
* Update data previously existing in the database.

## Key System Functionalities

### Setting Interests

* Setting Interests is a functionality provided when users create an account.
* The user may only set a maximum of 3 interests.

*Refer to the definition of Interests for more detail.*

### Pages and Desks

* + - * A single Group Home Page may have multiple Pages and each Page may have the following:
        + Desks
        + Past Posts
        + Page Information Display

### User Matching

* User Matching is a functionality occurring after user login where the app displays the icons of students who have similar interests and/or are in the same group(s) as the user.
* User Matching takes into account the Interests set by the user upon the sign-up process.
* User Matching results in
  + (i) user's being able to see suggested students that have a high similarity with the user's interests (User Matches)
  + (ii) suggested groups that are aligned with the user’s Interests (Top Groups).
* Two users are considered to have a high similarity when:
  + The user and the student have at least 2 of the same interests.
  + The user and the student have ranked their parameters in a similar order.

## Key UI Components

### Bars

* Main Navigation Bar — provides links to the user’s profile, message history, and friendship list.
* Sidebar — provides a view of the user’s joined groups (Joined Groups) and Friends Online display.
* Search Bar — allows users to search for groups and users related to their Interests.

### Containers

* User Matches Container — displays the icons of the user’s User Matches. The User Matches are obtained from the User Matching process.

### Pages

Each Group Home Page may contain multiple Pages.

Contained within a single Page are the following:

* + - * Page Information Display — a quick description of the Page.
      * Desks — allows members of a group to send and receive chat messages to and from fellow group members.
      * Past Posts Container — display past posts published to the group’s page.

### Desks

Contained within a single Desk are the following:

* Text Chat — chat message display sent to and from fellow group members along with the first name of each group member.
* Icon Display — display of the icons of each participant in the chat.
* Duration — text describing the duration Desk.
* Status — colored icon describing the status of the Desk, whether or not it is live or expired.
* Take a Seat Button — a button that allows group members to become participants of the Desk.

### Grids

* Top Groups Grid — displays top groups suggested to the user that they are not a member of. The groups are obtained from the User Matching process.
* Top Posts Grid — displays most upvoted posts in the user’s joined groups.

# Design Considerations

## Assumptions and Dependencies

* User has a stable internet connection
* Compatible with all browsers and operating systems

## End-user characteristics

* Creation, editing, and modification of posts
* Moderator has the ability to create, remove, and edit pages
* Moderator can remove user reported comments

## General Constraints and Limitations

* Time and resource limitations
  + Voice chat feature between users
* Security requirement
  + Authorization and password management
* Reliability of accurate matches between users
* All screens must be above 320px in order to meet usability requirements

## Goals and Guidelines

* Clean and simple UI
* Efficient and doesn’t use excessive memory
* Reliable and robust interface
* Documentation of all the code
* Protection of website against third party attacks (security of user data)

## Development Methods

The main software used for the design of the UI was Adobe XD. This method was selected due to the rapid prototyping features that it provides. It allows the pages to be distinct and resembling the final design.

* UML diagrams were used to visualize the interactions between users, moderators, and admin.
* Multiple revisions were made at each stage of the software development to ensure that the best possible outcome was achieved.
* SQA performed to ensure that all developers were on the same page and doing the desired task.

# Architectural Strategies

## Communication Mechanisms

The front-end and back-end of the application will communicate via RESTful API calls.

## Technology Specifications

### Back-end

Python

Due to time constraints and familiarity with Python, the back-end team will use Python as the primary language.

Django

Django comes with common libraries which are essential to building common functionalities like URL routing, authentication, an object-relational mapper (ORM), a templating system and db-schema migrations. sql

PostgreSQL

PostgreSQL was chosen as the database management system because it is a database suited for a production environment and also due to the fact that Heroku naturally supports PostgreSQL.

Heroku

The RESTful API will be hosted on Heroku.

### Front-end

HTML, CSS, JavaScript

HTML, CSS, and JavaScript are the standard technologies used to implement a web application.

Node.js

Node.js is the environment in which the front-end team will query and store data in the database.

Tailwind and PostCSS

Tailwind is a CSS framework useful for its easy setup and efficiency in building complex interfaces.

PostCSS is a tool built into Tailwind useful for its utility classes.

React JS and React Design Patterns

React JS is a JavaScript library to aid the front-end team in the creation of user interfaces for single-page applications (SPAs).

Redux

Redux is a state management tool that allows the front-end developers to monitor and manage the data contained within React components. In particular, Redux solves the problem of maintaining consistent state (data) between different React components like the Search Bar and the User Matches Container.

Vercel

Vercel will be used as the primary way to deploy the front-end of the StudySpace web application. Vercel is used due to its efficient software development workflow, Github integration, and familiarity; it is optimized for the front-end team.

### Miscellaneous

Git and Github

Git and Github are utilized as the main version control system.

Postman

Postman will be the primary way that API endpoints are tested.

## Front-end Goals

* The UI of the StudySpace web application will be hosted on Vercel as a SPA.
* The front-end of this application must be able to query and store data in the database using Node.js.

## Back-end Goals

* The back-end of this application will implement a RESTful API connected to a PostgreSQL database.

## Future Plans

The StudySpace web application developed for the project client is meant to be a proof-of-concept (see: Definitions) to demonstrate the team’s understanding of various software design principles and workflows.

## User Interface Paradigms

React JS

The main utility comes from easily creating reusable UI components. Each React component has its own system, state, and life cycles.

* Stateless Components — data is fetched from a database and important into the component.
* Conditional Rendering — allows the developer to create distinct components. One potential use for Conditional Rendering is the need to create different error messages for different sources of error.
* Render Props — reduces repetition by allowing components to dynamically render other components.

React aids the front-end developers in creating key reusable components such as the Search Bar and the User Matches Container.

React Redux

This library allows React components to read data from a global 'store' and allows components to dispatch actions to that store to update the state data.

## External Database Management and Persistence

The use of Heroku as the primary platform for deployment will resolve the problem of database persistence. Every time a new user joins the StudySpace web application, the database is updated and re-deployed to Heroku.

1. Design Decisions, Policies, and Tactics format

## Deployment

* The StudySpace website will be deployed using vercel, and will be coded in Javascript, HTML, CSS, PostCSS, and Tailwind. Backend is in Python/Django and the database will be done using PostgreSQL.

## Data Storage

* + - * Database will be on a server hosted by the StudySpace team.

## Coding Guidelines and Conventions

* Front end will be coded using Redux and React
* Main coding languages for front end are Javascript, CSS, and HTML
* Backend will be coded using Python and Django

## Ensuring Requirements Traceability

* Requirement Traceability will be ensured using a traceability matrix

## Software Testing

* Software is regularly tested by the StudySpace team after every update
  + Extensive debugging is done to ensure a smooth and reliable user experience
* Github used for version control and Postman used to test API endpoints
* White box testing handled by the StudySpace team until April 12, 2021

## Software Maintenance

* Software is maintained by the StudySpace team until April 12, 2021

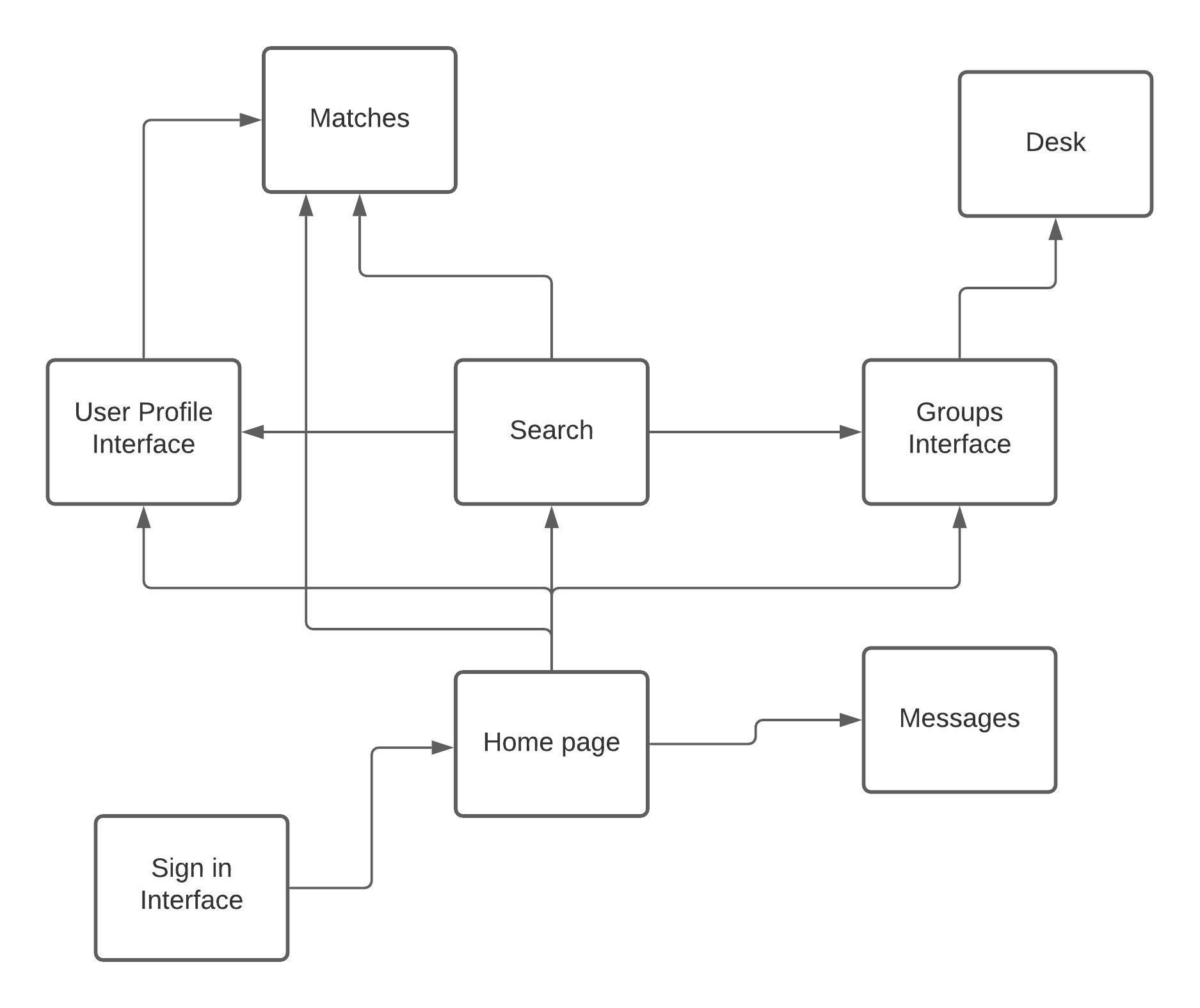
## File Structure Organization

* /host
  + Dotfiles: configuration files
* /components
  + React components
* /pages
  + Layout for different UI views
* /public
  + Static images
* /styles
  + Global styles

1. System Architecture

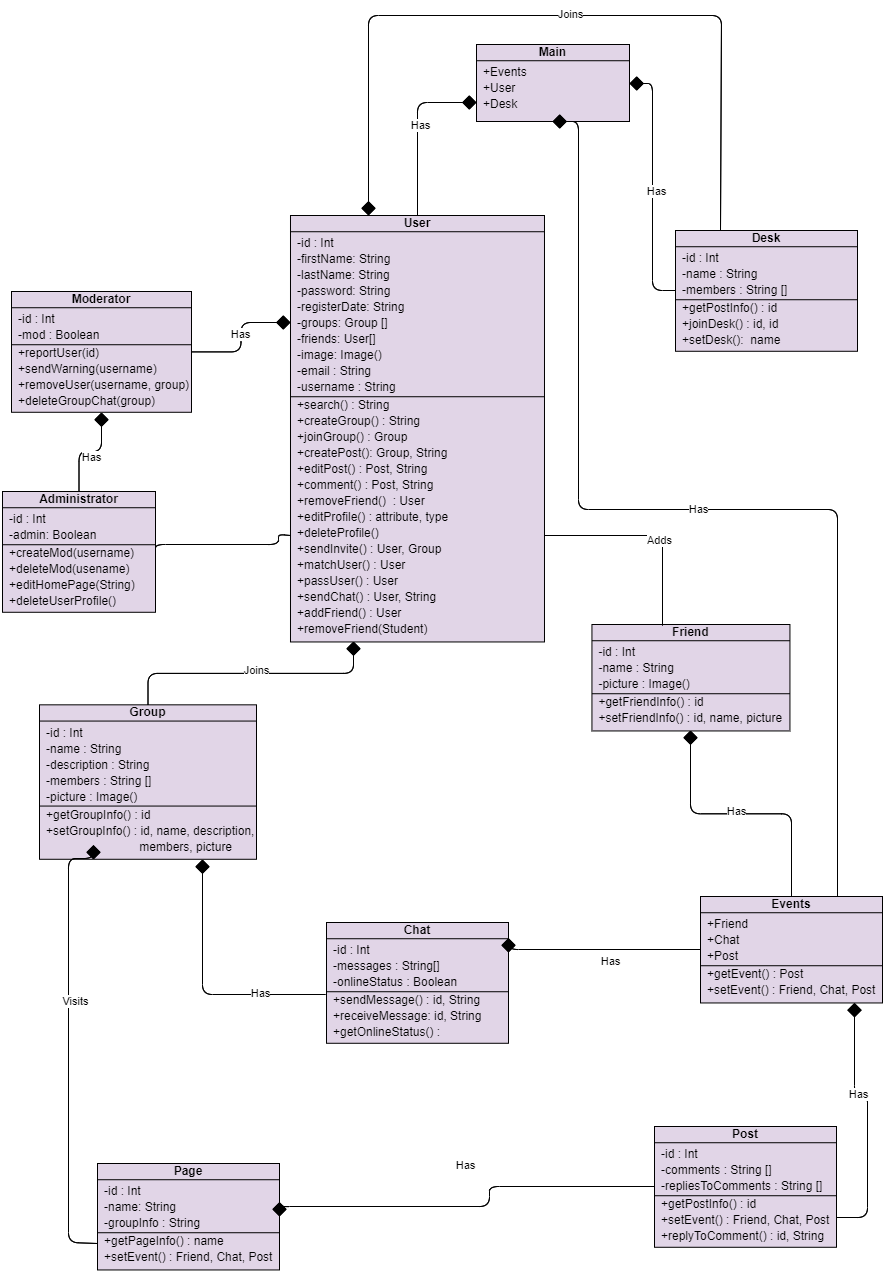
## System Architecture Description

* + - * Sign in Interface is the first page, where the user enters credentials to log into their account
      * Home page directs the user to other significant pages like Messages, User profile, Groups
      * Search can be used from the homepage to find Matches, Users, Groups, or Desks
  1. System Architecture Diagram

****

1. Architectural and Component-level Design

7.1 Class Diagram



7.1.1 Main Class

**Purpose**

The main class is the top-level of the program, which has methods to create Users, Desks, and Events.

**Prototype:** Main()

**Called By**

* N/A (This is the top-level of the program)

**Calls**

* User
* Desk
* Events

**Algorithm**

Creates User, Desk, and Event objects as needed.

7.1.2 User Class

**Purpose**

Contains attributes and methods possessed by a base level User. All methods and attributes are inherited by Moderator and Administrator classes. This Class is intended for Students of an Ontario University. Users have basic privileges that allow them to access and create Desks, Groups, Posts, and send/receive Chats. Moderators and Administrators possess higher-level permissions that general users don’t have, including the ability to manage groups.

**Prototype**: Public User(id, username, password, email, firstName, lastName, registerDate, groups(), friends(), Image())

groups(), friends(), and Image() may be left as NULL and updated later.

**Inputs:**

* Id – Unique ID number assigned to each User object by class constructor.
* Username – Must be unique, self-determined by the user.
* Password – Used alongside username for authentication.
* Email – Email address
* firstName – First name
* lastName – Last name.
* registerDate – Assigned by constructor
* groups() – List of groups that the user is a part of
* friends() – List of friends that the user has
* Image() – Profile picture image file

**Outputs**

* User object

**Restrictions**

· id, username, password, email, firstName, lastName are all required inputs.

**Called by**

· Main.User()

**Algorithm**

· Creates a User object provided the required inputs.

7.1.3 Moderator Class

**Purpose**

As a subclass of User, Moderator inherits all methods and attributes of a general User, but has increased permission. Increased permissions allow the Moderator of a group to prevent rule breaking by sending warnings and removing problematic users and posts from the group.

**Prototype**: Public Moderator(id, Boolean)

**Inputs:**

* Id – Unique ID number assigned to each User object by class constructor.
* Boolean – True to set Moderator, False to demote Moderator into a user

**Outputs**

* None

**Restrictions**

* None

**Called by**

* Administrator

**Algorithm**

· Promotes or creates a user and assigns increased privileges to them.

7.1.4 Administrator Class

**Purpose**

As a subclass of Moderator, Administrator inherits all methods and attributes of Users and Moderators and adds additional privileges, including assigning and removing group moderators, editing the home page, and deleting user accounts.

**Prototype**: Public Admin(id, Boolean)

**Inputs:**

* Id – Unique ID number assigned to each User object by class constructor.
* Boolean – True to set Admin, False to demote Admin into a Moderator.

**Outputs**

* None

**Restrictions**

* None

**Called by**

* Administrator

**Algorithm**

* Promotes or creates a user and gives admin privileges.

7.1.5 Friend Class

**Purpose**

Store and maintain friendship data between users.

**Prototype**: Public Friend(id, name, Picture)

**Inputs:**

* Id – Id of the user that has sent the friend request.
* Name – Name of the user who is being requested as a friend.
* Picture – Profile picture of the user who is being requested as a friend.

**Outputs**

* None

**Restrictions**

* None

**Called by**

* User

**Algorithm**

* Creates a friend object.
* Adds friend to friends list attribute in user profile.

7.1.6 Group Class

**Purpose**

A Group is a collection of two or more users who have connected through StudySpace either by Invitation or a User Match. All members of a Group automatically have access to a group-wide Chat upon joining.

**Prototype**: Public Group(id, name, description)

**Inputs:**

* Id – Unique ID number assigned to each Group object by class constructor.
* Name – Name of Group assigned by the creator of the group.
* Description – Description of purpose of the Group and/or what its about.

**Outputs**

* None

**Restrictions**

* None

**Called by**

* User
* Moderator
* Administrator

**Algorithm**

* Creates a group and assigns a unique id.
* Upon joining a group, Users are able to view and interact with posts in the group as well as send/receive chats amongst all group members via the group-wide chat log.

7.1.7 Page Class

**Purpose**

A Page is where Posts are viewed and published. Each Group has at least one page.

**Prototype**: Public Page(id, name, groupInfo)

**Inputs:**

* Id – Unique ID assigned to page by class constructor.
* Name – Name of page
* groupInfo - Information about the group which the page belongs to, including the groupID, name, members, and picture.

**Outputs**

* None

**Restrictions**

* None

**Called by**

* User
* Moderator
* Administrator

**Algorithm**

* Creates a page and assigns unique id.
* Assigns a page to a group.

7.1.8 Desk Class

**Purpose**

A Desk is a virtual space where students may collaborate. Each desk is temporary; it only exists so long as there are people using it. After everyone leaves the desk, it is purged from memory. Each desk has a Moderator who can set a limit to the number of members and add signifiers such as language, age category, and other factors. Users can sort desks based on their preferences or join a random one.

**Prototype**: Public Desk(id, name, members)

**Inputs:**

* Id – Unique ID assigned to desk by class constructor.
* Name – Name of desk
* Members - List of users that are members.

**Outputs**

* A new desk

**Restrictions**

* None

**Called by**

* User
* Moderator
* Administrator

**Algorithm**

* Creates a desk and assigns unique id.
* Assigns the desk to a group.

7.1.9 Post Class

A Post is an object which can include photos and text. Users create Posts within Groups for all other members of that group to see. Users can make a Post about anything they wish, including buying/selling textbooks, searching for a tutor, asking questions, looking for advice, etc. Users can write a Response to a Post. The authors of a post can edit their post or take it down at any time.

**Prototype**: Public Post(id, comments, repliesToComments).

**Inputs:**

* Id – Unique id assigned to post by constructor.
* Comments – Comment section Users’ responses to the post..
* repliesToComments – Each comment on a post has its own list of replies.

**Outputs**

* Post object.

**Restrictions**

* None

**Called by**

* User
* Moderator
* Administrator

**Algorithm**

* Create a post object.
* Assigns the post to a page.

7.1.10 Events Class

**Purpose**

Events are scheduled meetings for members of a group. A scheduled meeting is a time when all members of the group are to go online and engage in a group-wide discussion in the chat log.

**Prototype**: Public Events(Friend, Chat, Post).

**Inputs:**

* Friend – Which friends are going to the event.
* Chat – The chat log in which the meeting is held.
* Post – A post in the group for which the event is scheduled. The post outlines what the purpose of the event is and when it will be.

**Outputs**

* Scheduled event.

**Restrictions**

* None

**Called by**

* User
* Moderator
* Administrator

**Algorithm**

* Creates and schedules an event.

7.1.11 Chat Class

**Purpose**

Chat is a feature where all members of a Group can send text messages to each other. Two users can also send direct chats to one another if they are friends. The Chat log is viewed through a chat box on the Group’s page, or through the user’s main page in the case of direct chat conversations.

**Prototype**: Private Chat(id, messages).

**Inputs:**

* Id – Id of the friend a user intends to chat with, or, in the case of group-wide chats, the groupId of the group which the chat log belongs to.
* Messages – The content of the text messages exchanged within the chat log.
* OnlineStatus – Indicates whether or not the user(s) in the chat log are online.

**Outputs**

* Chat messages exchanged between users.

**Restrictions**

* None

**Called by**

* User
* Moderator
* Administrator

**Algorithm**

* Creates a new chat log upon the creation of a group, or initialization of friendship in the case of direct chats between friends.
* Calls send and receive methods to create and get messages between users.

# Data Design

## Data Structures

### Global data structure

The database will be used globally to retrieve information regarding many components of StudySpace. Each table in the database has its own respective class. The classes representing the data from the database will also be used globally.

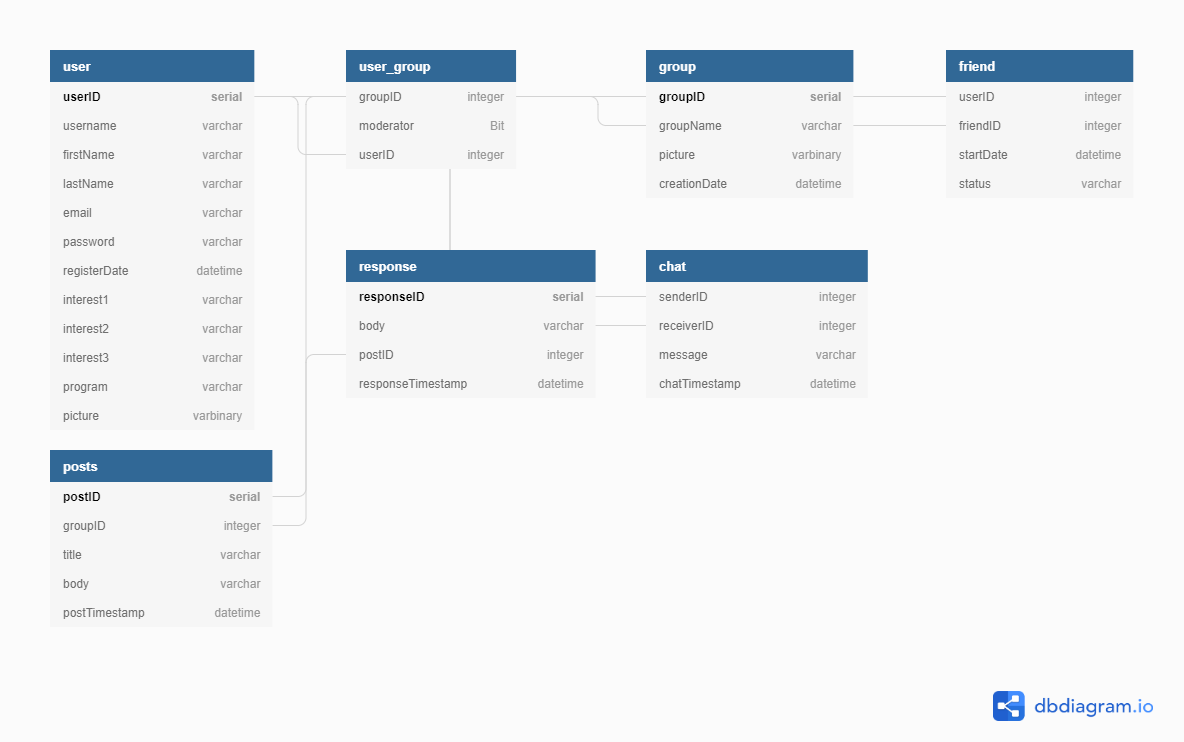
### Temporary data structure

Data structures for and regarding the DESK feature will be all temporary. Since each DESK instance is only temporary; the data structures related to the DESK feature will also be temporary. As soon as people stop using the DESK, every file/data structure will be deleted.

### Database description

For the database, we start with the user’s general information and assign the user a unique ID. With a user’s ID we will be able to link the user’s friends, preferences, groups the user is part of to the user. To keep track of a user’s private messages with other users, we would also need a table keeping track of messages between any given two users. A post table and a response table would be able to keep track of posts made by a group as well as the comments made on the post.

## Database Diagram

****

## Database tables

### user Table

* userID

A unique ID given to all users who sign up for StudySpace.

* username

The username in which the user is referred to on the app.

* firstName

The first name of the user.

* lastName

The last name of the user.

* email

The email address of the user.

* password

The password of the user.

* registerDate

The date the user created an account on StudySpace.

* interest1

An interest of the user used for generating User Matches.

* interest2

An interest of the user used for generating User Matches.

* interest3

An interest of the user used for generating User Matches.

* program

The academic program which the user is enrolled in.

* picture

A picture that the user can upload for their display picture.

### group Table

* groupID

A unique ID given to a group when it is created.

* groupName

The name of the group.

* picture

A picture that a group member uploaded for the group's display picture.

* creationDate

The date the group was created.

### user\_group Table

* groupID

The unique ID of the group.

* moderator

Boolean value representing whether the user is a moderator.

* userID

The unique ID of the user that is in the group.

### friend Table

* userID

The unique ID of a user.

* friendID

The unique id of a user who is friends with the user associated with user\_id.

* startDate

The date on which the two users with user IDs matching user\_id and friend\_id became friends on the platform.

* status

There are 3 possible statuses of a friend request:

* Send friend request (not yet friends)
* Pending friend request (request sent but not accepted)
* Friends (request accepted; users are now friends)

### chat Table

* senderID

The unique ID of the user sending the message.

* receiverID

The unique ID of the user receiving the message.

* Message

The text of a chat message.

* chatTimestamp

The date and time the message was sent.

### posts Table

* postID

The unique ID of the post.

* groupID

The unique ID of the group that uploaded the post.

* title

The title of the post.

* body

The actual body/message of the post.

* postTimestamp

The date and time the post was posted.

### response Table

* responseID

The unique ID of the response on a given post.

* body

The body/message of the response.

* postID

The unique ID of the post.

* responseTimestamp

The date and time the response was posted.

1. User Interface Design

## User Interface Design

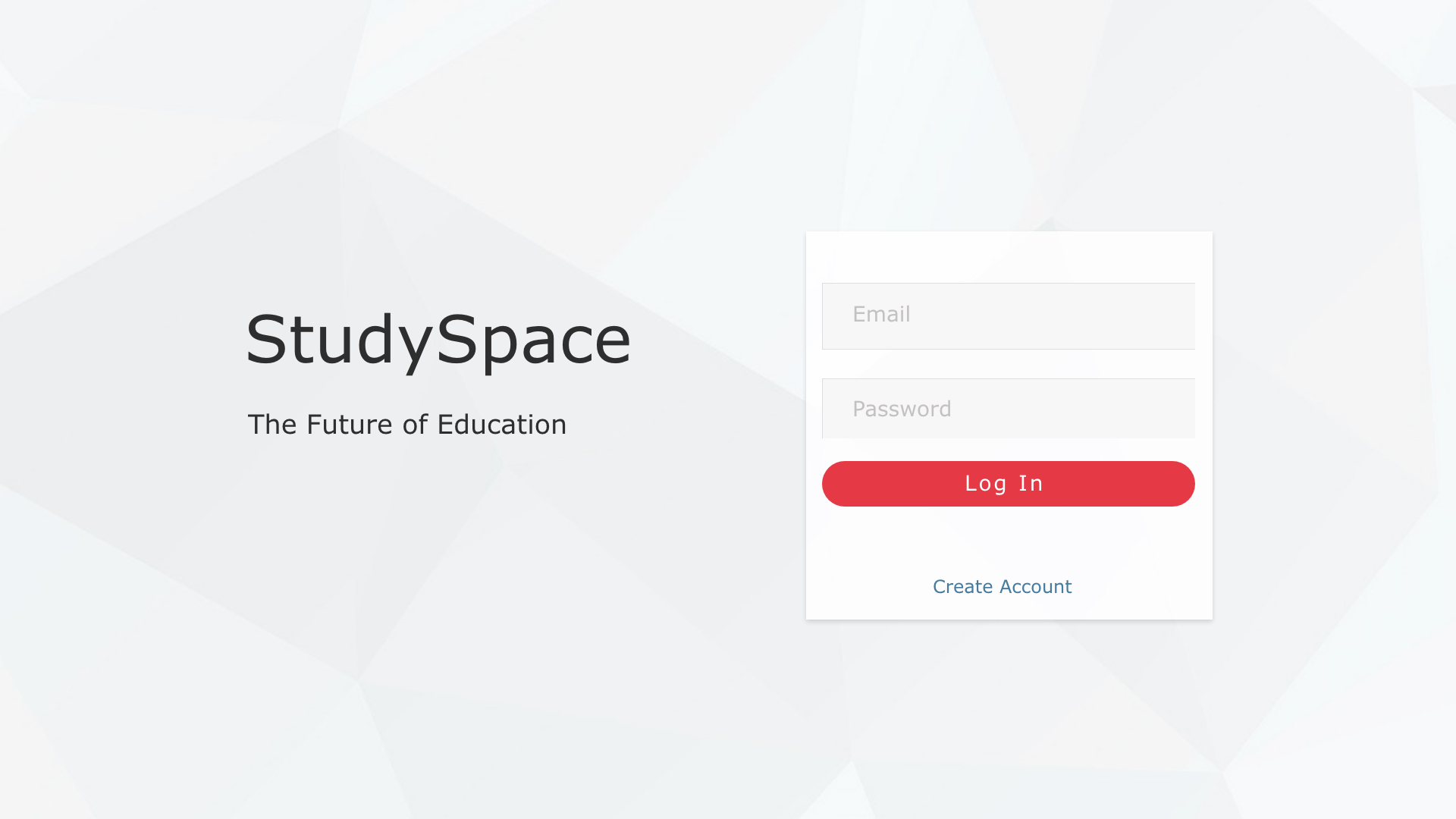
* + 1. Color Guidelines

|  |  |  |
| --- | --- | --- |
| Colour | Hex Value | Use |
|  | #1D3557 | * Header |
|  | #F5F5F5 | * Background behind text and images |
|  | #E63946 | * Buttons |
|  | #FFFFFF | * Background of page * Text over dark background |
|  | #2E2E2E | * Text over light background |
|  | #C3C1C1 | * Text in a textbox |
|  | #457B9D | * Link |

* + 1. Font Guidelines
       - We will use default fonts as provided by Tailwind CSS
         * San Francisco

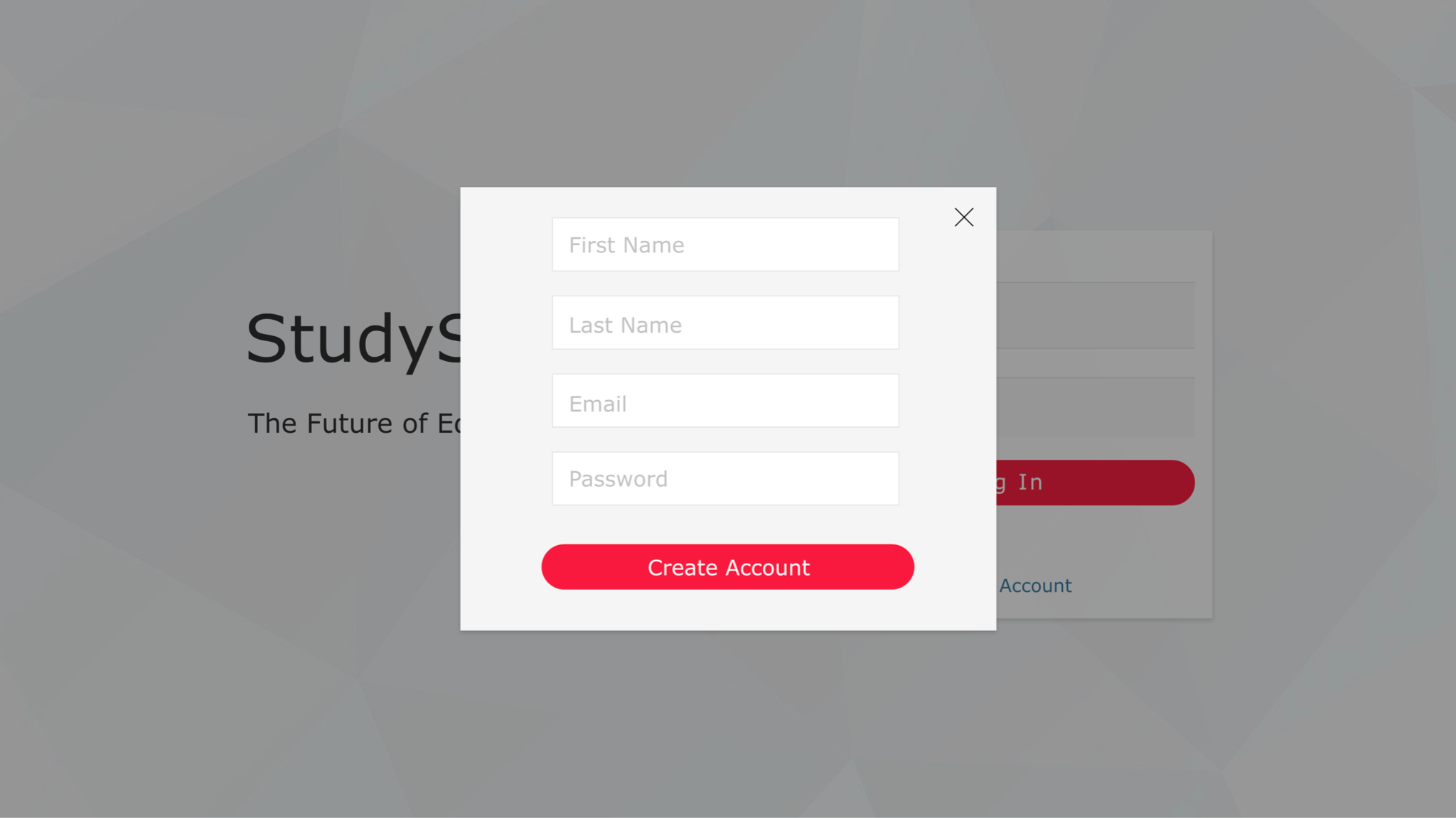
## User Interface Views

* + 1. Landing Page (logged out)



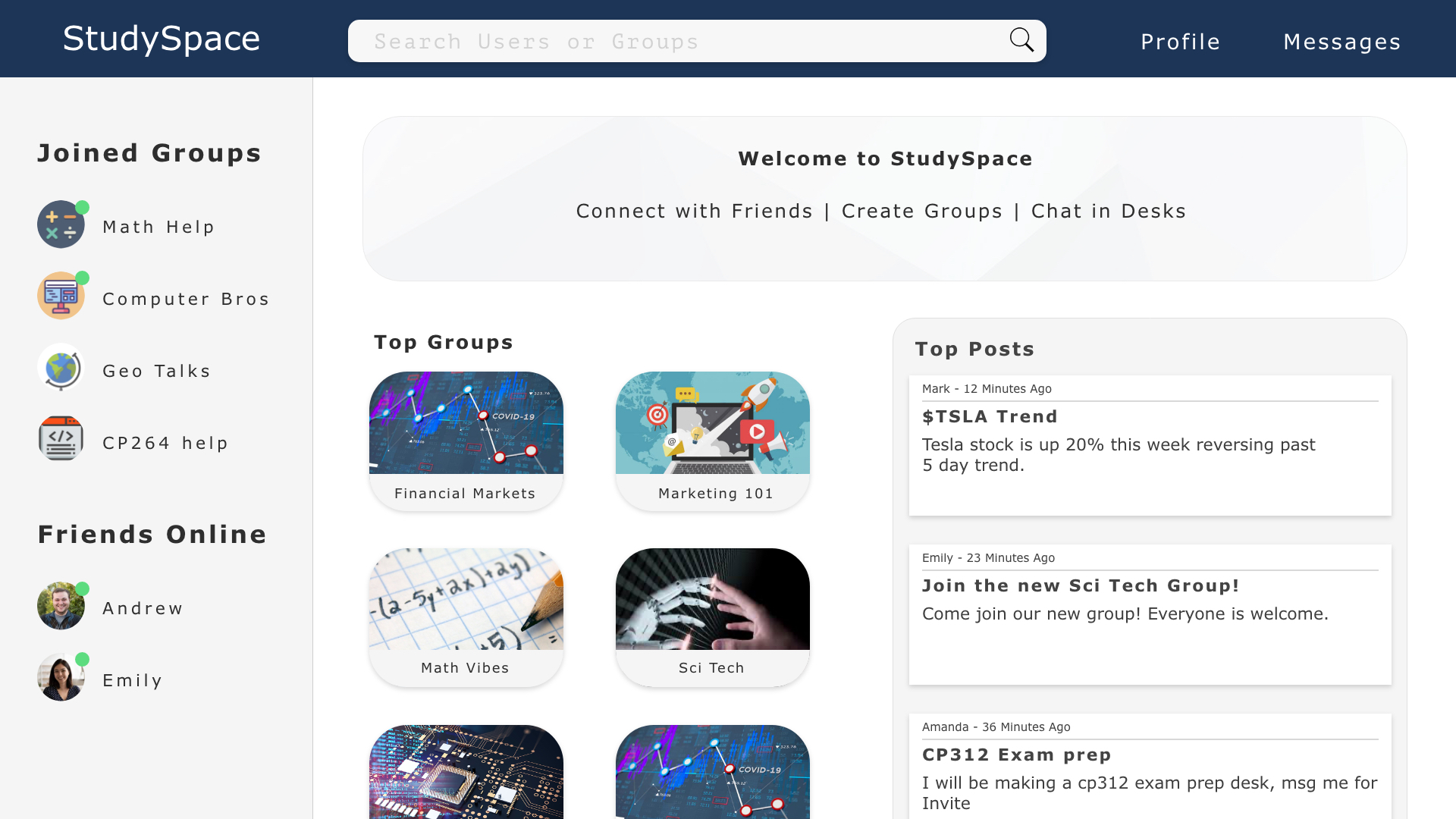
|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| Email | Text input | Enter user email |
| Password | Text input | Input user password |
| Log In | Button | Links to logged in Home Screen |
| Create Account | Link | Links to Sign Up page |

* + 1. Sign Up View



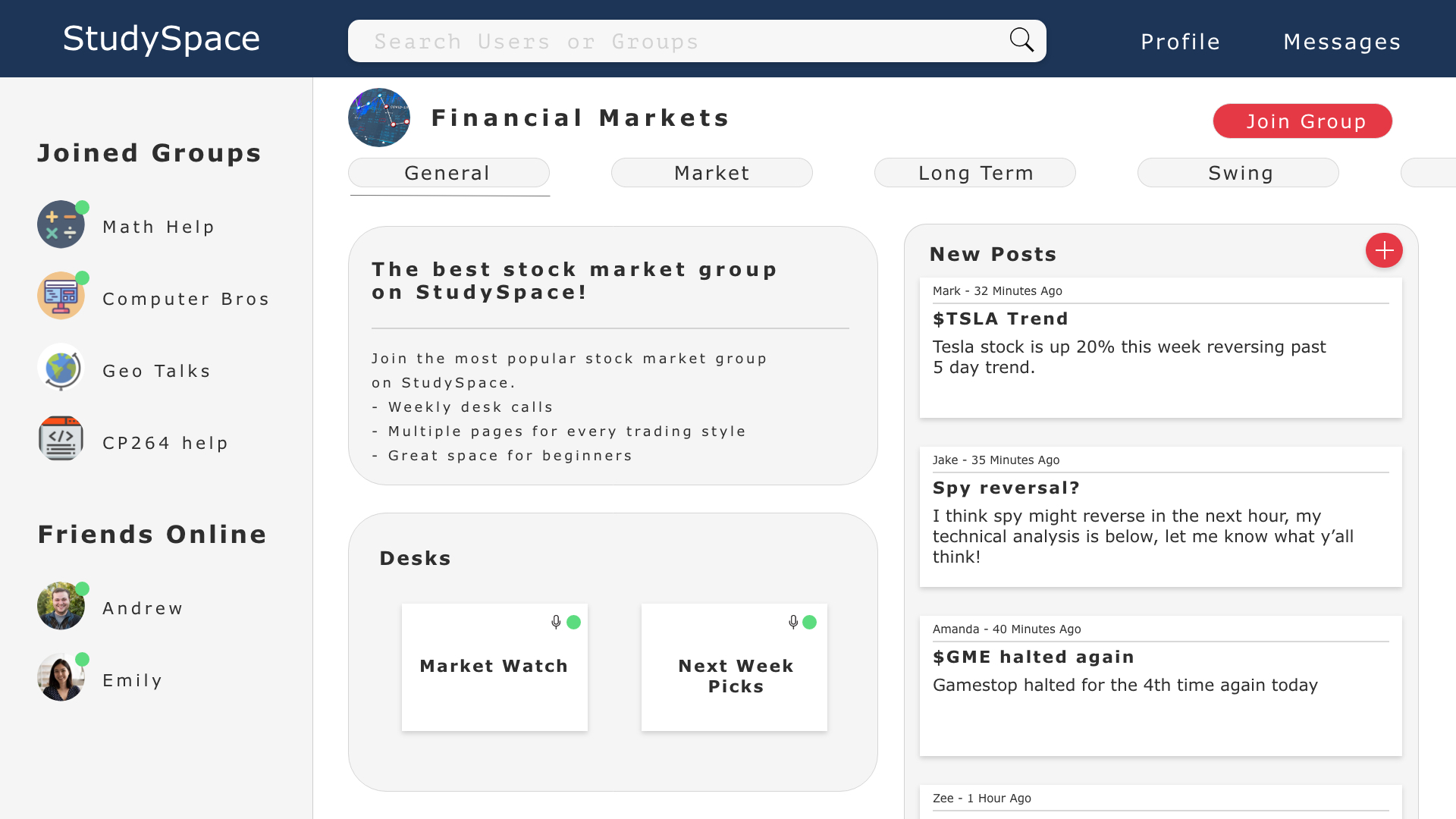
|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| First Name | Text input | Input your first name |
| Last Name | Text input | Input your last name |
| Email | Text input | Input user email (needed to log in) |
| Password | Text input | Input unique password |
| Create Account | Button | Confirms the account, user details are then stored in database |

* + 1. Home Page (Logged In)



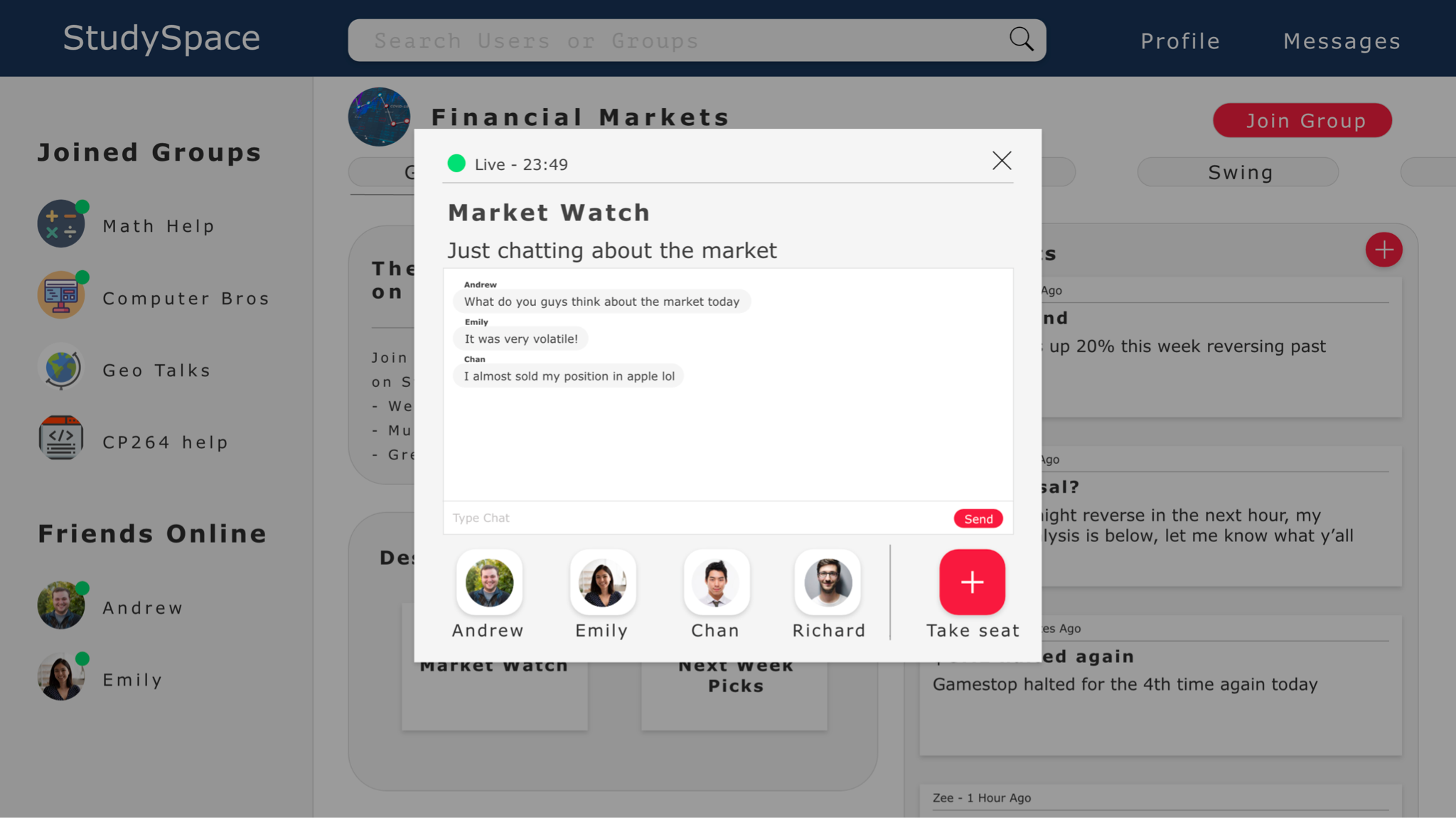
|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| Search Bar | Text input | Allows user to search for users or groups on the site as well as filter through users using their interests |
| Profile | Button | Links to profile page |
| Messages | Button | Links to users messages |
| Your Groups | Sidebar menu | List of links to users joined groups |
| Friends Online | Sidebar menu | List of users friends currently online |
| Top Groups | Widget | Links of popular groups based on user preferences |
| Top Posts | Widget | Links of popular posts based on user preferences |

* + 1. Groups Home Page



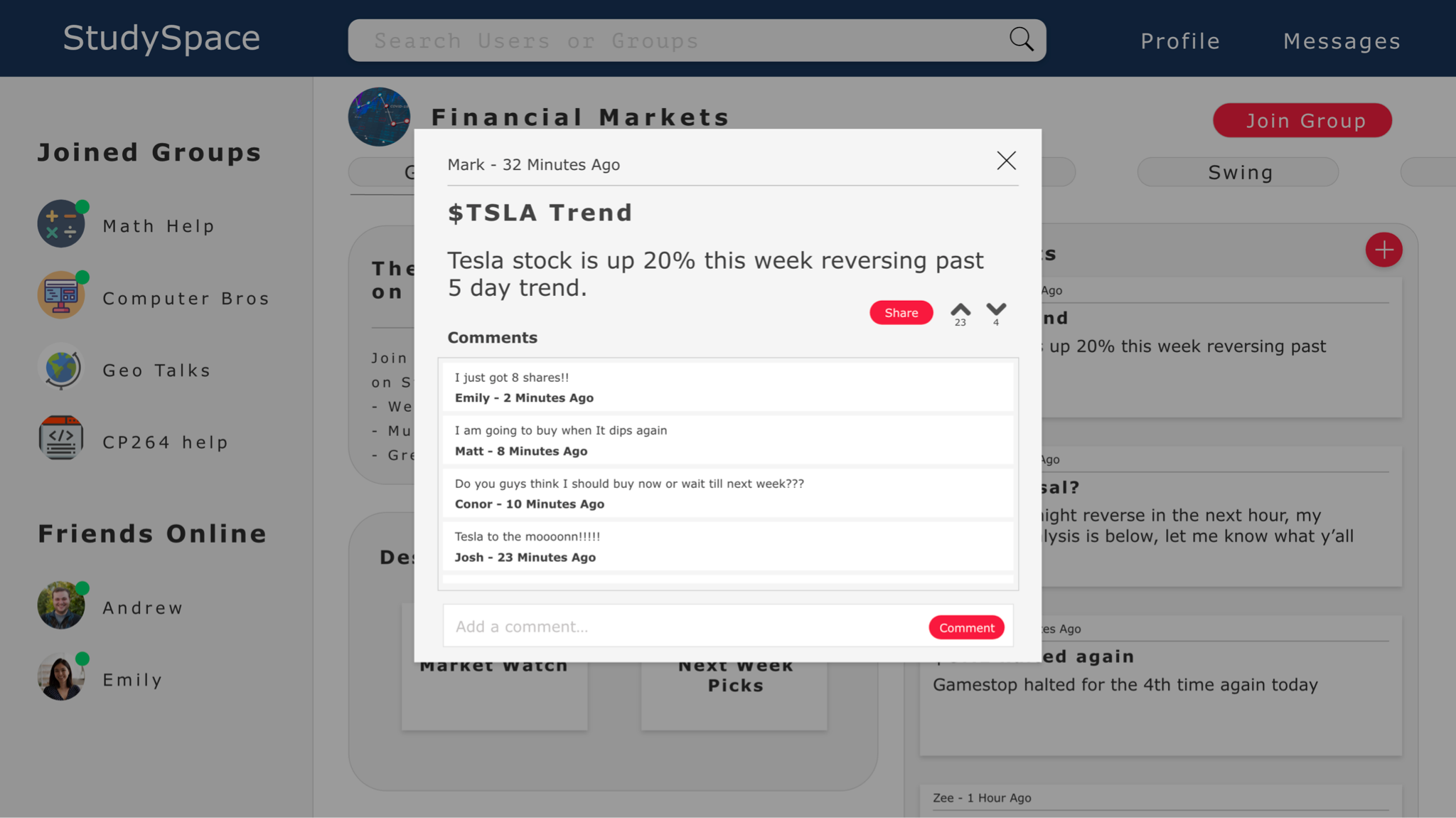
|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| General | Page | Current page of the group the user is viewing. Multiple pages available |
| Join Group | Button | Allows user to join the group, it will save to the “Your Groups” sidebar once joined |
| Desks | Widget | Links all live desks available to join |
| New Posts | Widget | Links posts by newest on top (within 24 hours) |

* + 1. Desk View



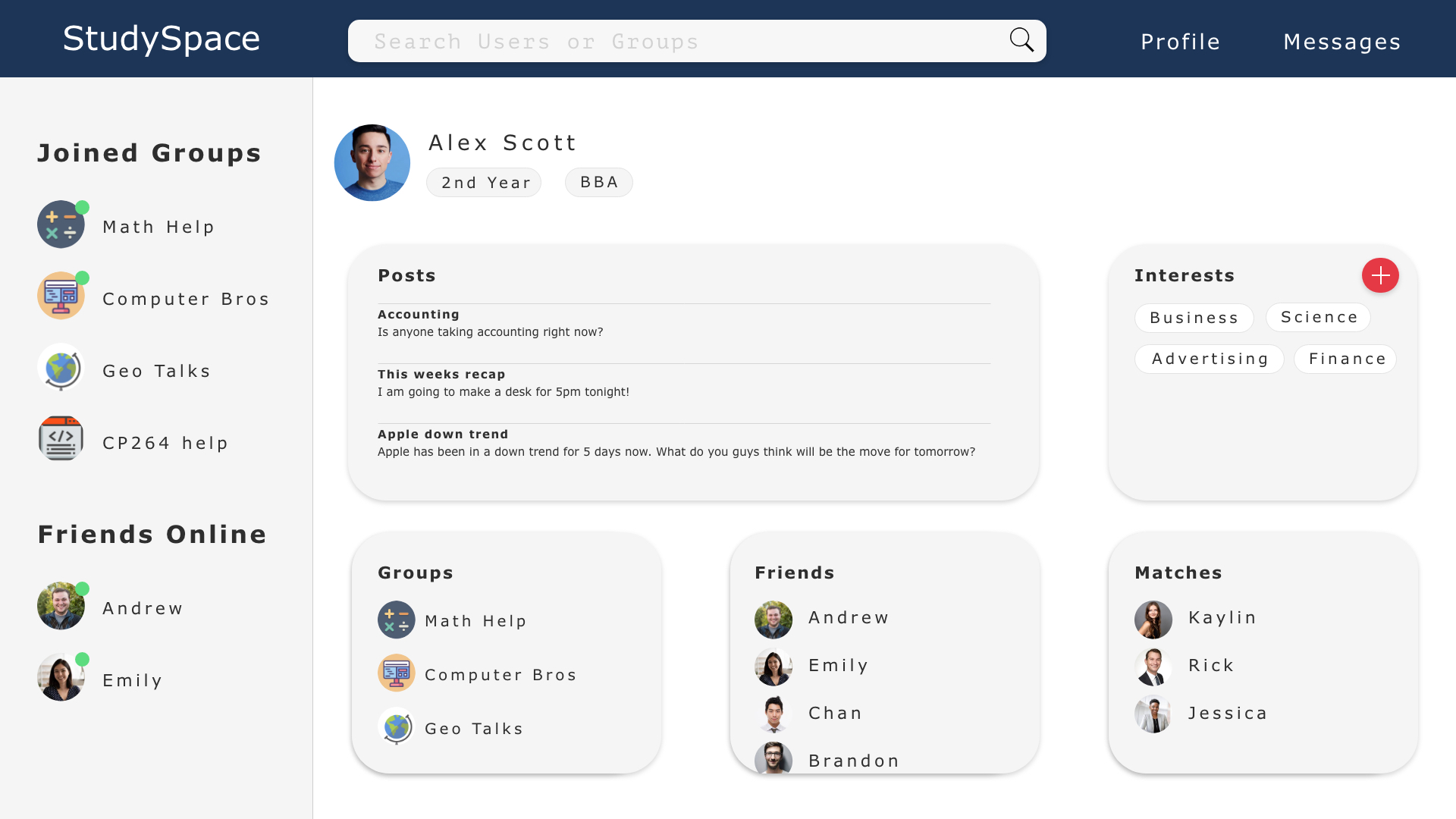
|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| Take seat | Button | Allows user to join the desk |
| Type Chat | Text input | Allows user to chat with other users in the desk |

* + 1. Posts View



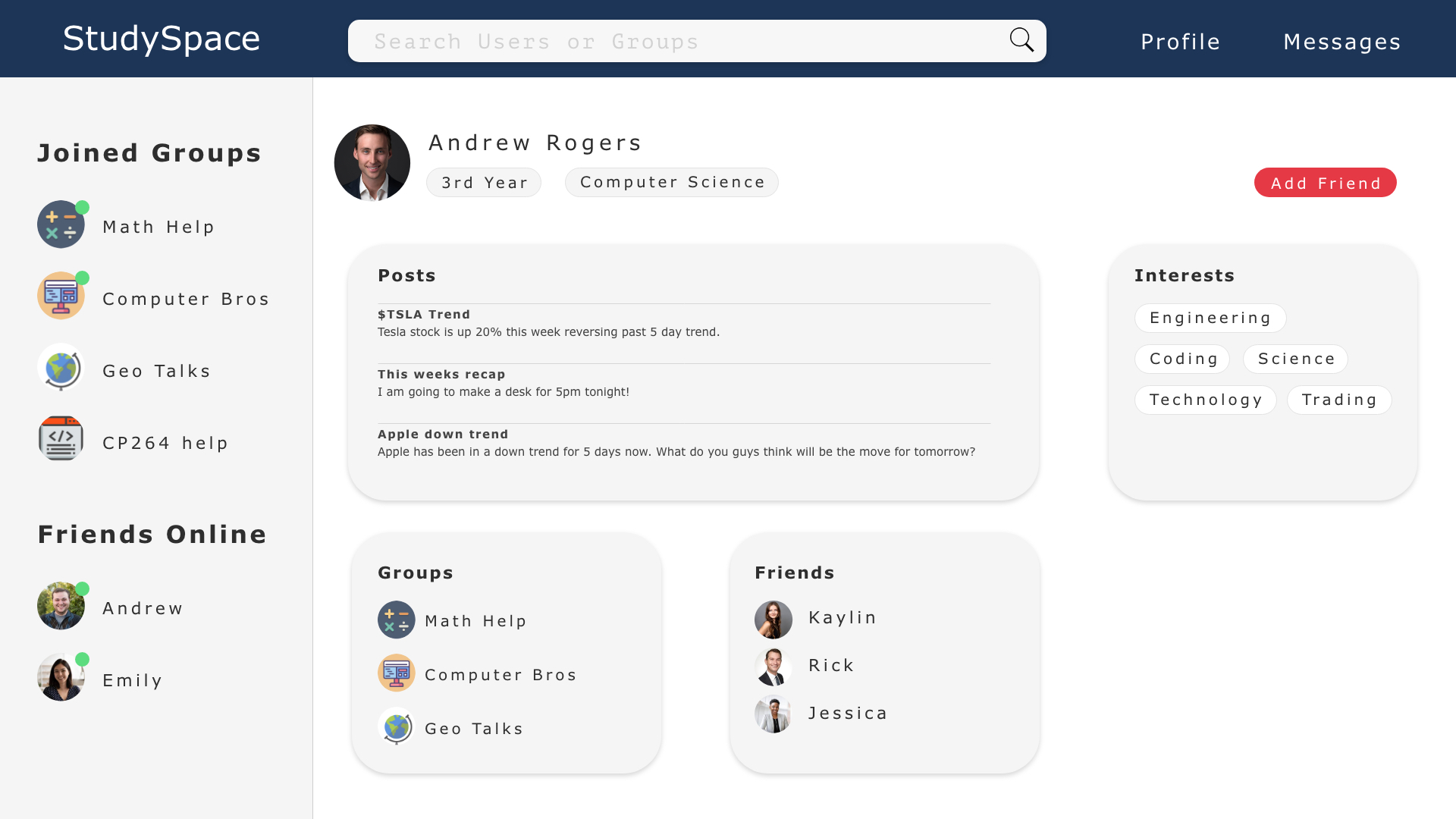
|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| Share | Button | Allows user to send the post via link to another user |
| Upvote | Button | Allows user to share their opinion using upvote if they like the post |
| Downvote | Button | Allows user to share their opinion using downvote if they dislike the post |
| Add a comment | Text input | Textbox for user to comment under post |
| Comment | Button | Allows user to publish comment |

* + 1. User Profile Page



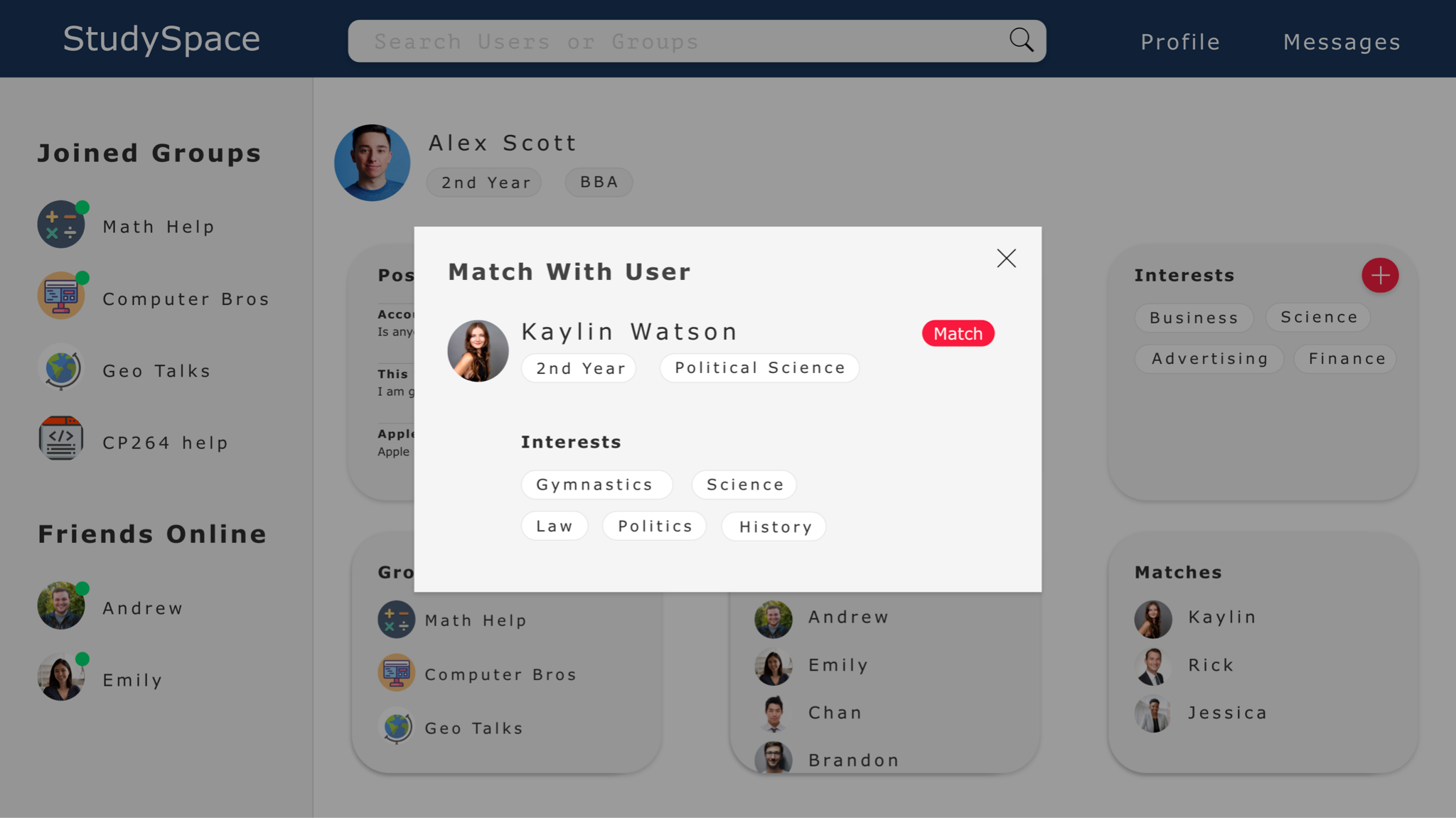
|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| Posts | Widget | Displays the posts written by user |
| Interests | Widget | Allows users to display their interests as well as add new ones via the plus button |
| Groups | Widget | Displays all of the user’s joined groups |
| Friends | Widget | Displays all of the user’s friends |
| Matches | Widget | Displays other users who have matched with the user based on similar interests |

* + 1. Profile Page (another user)



|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| Posts | Widget | Allows user to view other users posts |
| Interests | Widget | Allows user to view other users specific interests |
| Groups | Widget | Allows user to view other users joined groups |
| Friends | Widget | Allows user to view other users friends |

* + 1. User Match View



|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| Interests | Widget | Allows user to view other users specific interests |
| Match | Button | Allows user to confirm the match if wanted |

1. Data Dictionary

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Data Type** | **Description** | **Constraints** |
| changeTitle() | Void | Change title on each page |  |
| newUser() | Void | Creates a new user |  |
| editUser() | Void | Edits a user |  |
| editPage() | Void | Edits page |  |
| deleteUser() | Void | Delete a user |  |
| deletePage() | Void | Delete page |  |
| email() | Varchar | Individual emails of users | Valid email address |
| editUserPass() | Varchar | Password recovery |  |
| title() | Varchar | Title of a page | Must be at least 3 characters long and cannot start with numbers or symbols |
| retrieveTitle() | Varchar | Get the title of a page |  |
| retrieveEmail() | Varchar | Get the email of a user |  |
| username() | Varchar | Username |  |
| firstName() | Varchar | First name of a user | Must be at least 3 characters long and cannot include numbers or symbols |
| lastName() | Varchar | Last name of a user | Must be at least 3 characters long and cannot include numbers or symbols |
| password() | Varchar | User password | Contains at least 8 characters, one symbol, and one number |
| registerDate() | datetime | The date of account creation | Must follow Canadian date format (YYYY - MM - DD) |
| joinGroup() | Void | Used to join a group |  |
| createPost() | Void | creates a new post |  |
| editPost() | Void | Edits a post |  |
| userID() | Serial | Unique ID given to all users who sign up for StudySpace.  username | Numeric value |
| interest1() | Varchar | Interest of user |  |
| interest2() | Varchar | Interest of user |  |
| interest3() | Varchar | Interest of user |  |
| program() | Varchar | The program |  |
| picture() | Varbinary | Image |  |
| postID() | Serial | Unique ID of the post. | Numeric Value |
| groupID() | Integer | Unique ID given to a group when it is created. | Numeric Value |
| body() | Varchar | Actual body/message of the post. |  |
| postTimestamp() | Datetime | The time the post was posted | 12 hour clock format |
| moderator() | Bit | Boolean value representing whether the user is a moderator. |  |
| responseID() | Serial | ID of a response | Numeric Value |
| responseTimestamp() | Datetime | The time the response was posted. | 12 hour clock format |
| groupName() | Varchar | Name of a group | Must be at least 3 characters long and cannot start with a number or symbol |
| creationDate() | Datetime | Date of group creation | Must follow Canadian date format (YYYY - MM - DD) |
| senderID() | Integer | Unique ID of the user sending the message. | Numeric Value |
| receiverID() | Integer | Unique ID of the user receiving the message. | Numeric Value |
| message() | Varchar | A message |  |
| chatTimestamp() | Datetime | Time of a chat | 12 hour clock format |
| friendID() | integer | Unique id of a user who is friends with the user associated with user\_id. | Numeric Value |
| startDate() | Datetime | The date on which the two users with user IDs matching user\_id and friend\_id became friends on the platform. | Must follow Canadian date format (YYYY - MM - DD) |
| status() | Varchar | Status of a friend request |  |